Date⇒ 17-02-2021
Module⇒ Backend
Lecture By⇒ Akash Handa
Subject ⇒ Redis

What is Redis?

Redis is an in-memory data structure store, used as a distributed, in-memory key–value database, cache and message broker, with optional durability. Redis supports different kinds of abstract data structures, such as strings, lists, maps, sets, sorted sets, HyperLogLogs, bitmaps, streams, and spatial indexes.

What is REDIS?

Developed in 2009, REmote Dictionary Server (REDIS) is an open source, NoSql key value database. You can say it's a data structure server for developers to organize and use data efficiently and quickly. Redis allows the user to store vast amounts of data without the limitation of a relational database unlike MongoDB, MySQL, etc. It is written in ANSI C and runs on POSIX like your Macintosh.

Why use REDIS?

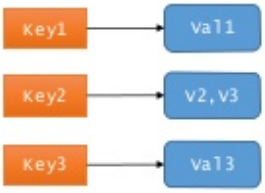
It is used for cache management and speeding up the web application by using a structured way to store data in the memory. It is, therefore, faster than conventional database techniques like MySQL, MongoDB, and Oracle.

Redis uses key value storage techniques, that is, every data structure is represented as a key. Redis has a number of keys to represent as many formats, resulting in more operations from the user perspective and reduced load from the client perspective.

Unlike MongoDb, which is a disk-based data storage, Redis holds all its database in memory, using disk only persistence technique, which stores the data in computer RAM, thereby making the processing extremely fast. It uses a memory caching technique which allows users to store data in a more durable and robust manner.

Redis supports the following Data structures. Regardless of their type, they are accessed by a key.

What is redis



- Redis is a **Key Value NoSQL database**
- **Open source** (BSD licensed)
- **In memory** data structure store. All data is served from memory
  - Redis mantra - data served from memory, disk used for storage
- Offers **high performance**, replication, and a unique data model
- Supports **five different data structures** - strings, lists, sets, hashes, sorted sets (as value) – redis is also called **data structure server**
- Used as database, **cache** and message broker.
- Actually stands for **RE**mote **DI**ctionary **S**erver
- Redis is often compared to memcached, which is a very high performance, key-value cache server.
- Supports built in replication, Lua scripting, on disk persistence, limited transaction
- Written in ANSI C, supports multiple platform

Redis Commands

Redis Commands - Lists & Hashes

Lists	
Push on either end *redis> RPUSH jobs "foo"* *(integer) 1* *redis> LPUSH jobs "bar"* *(integer) 1*	RPUSH/LPUSH [key value] O(1)
Pop from either end *redis> RPOP jobs* *"foo"* *redis> LPOP jobs* *"bar"*	RPOP/LPOP [key] O(1)
Blocking Pop *redis> BLPOP jobs* *redis> BRPOP jobs*	BRPOP/BLPOP [key] O(1)
Pop and Push to another list *redis> RPOPLPUSH jobs proc* *"foo"*	RPOPLPUSH [src dst] O(1)
Get an element by index *redis> LINDEX jobs 1* *"foo"*	LINDEX [key index] O(N)
Get a range of elements *redis> LRANGE jobs 0 -1* *1. "bar"* *2. "foo"*	LRANGE [key start stop] O(N)

Hashes	
Set a hashed value *redis> HSET user:1 name John* *(integer) 1*	HSET [key field value] O(1)
Set multiple fields *redis> HMSET user:1 lastname Smith visits 1* *OK*	HMSET [key field value ...] O(1)
Get a hashed value *redis> HGET user:1 name* *"John"*	HGET [key field] O(1)
Get all the values in a hash *redis> HGETALL user:1* *1) "name"* *2) "John"* *3) "lastname"* *4) "Smith"* *5) "visits"* *6) "1"*	HGETALL [key] O(N) : N=size of hash.
Increment a hashed value *redis> HINCRBY user:1 visits 1* *(integer) 2*	HINCRBY [key field incr] O(1)

Redis With Node

```
let express = require('express');
let axios = require('axios');
let redis = require('redis');
let app = express();
let port = 8777;

const client = redis.createClient({
    host:'localhost',
    port:6379
})

app.get('/data',(req,res) => {
    const userinput = (req.query.country).trim();
    const url =
`https://en.wikipedia.org/w/api.php?action=parse&format=json&section=0&pag
e=${userinput}`;

    // check in redis
    return client.get(`wiki:${userinput}`,(err,result) => {
        if(result){
            const output = JSON.parse(result);
            //return res as found in redis
            return res.send(output)
        }else{
            return axios.get(url)
                .then(response => {
                    //got response form api
                    const output = response.data;
                    // save in redis

client.setex(`wiki:${userinput}`,3600,JSON.stringify({source:'Redis',outpu
t}))
                    //return api response as data was not in redis
                    res.send({source:'api',output})
                })
        }
```

```
    })

})

app.listen(port,(err) => {
    console.log(`Server is running on port ${port}`)
})
```

Redis with Mongodb

```
let express = require('express');
let mongo = require('mongodb');
let mongodb = mongo.MongoClient;
let redis = require('redis');
let app = express();
let port = 8777;
const url = "mongodb://localhost:27017"

const client = redis.createClient({
    host:'localhost',
    port:6379
})

app.get('/data',(req,res) => {
    const userid = (req.query.id);

    // check in redis
    return client.get(`uid:${userid}`,(err,result) => {
        if(result){
            const output = JSON.parse(result);
            //return res as found in redis
            return res.send(output)
        }else{
            mongodb.connect(url,(err,connection)=> {
                const dbo = connection.db('aryalogin');
                let id = mongo.ObjectID(userid);
                dbo.collection('users').findOne({_id:id},(err,data)=>{
                    const output = data;
                    // store data in redis
```

```
client.setex(`uid:${userid}`,3600,JSON.stringify({source:'Redis',output}))
                    return res.send({source:'mongo',output})
                })
            })
        }
    })

})

app.listen(port,(err) => {
    console.log(`Server is running on port ${port}`)
})
```